View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Elsevier - Publisher Connector

J. Symbolic Computation (2002) **34**, 145–157
doi:10.1006/jsco.2002.0547
Available online at http://www.idealibrary.com on IDEAL®

Interval Arithmetic in Cylindrical Algebraic Decomposition

GEORGE E. COLLINS†, JEREMY R. JOHNSON‡
AND WERNER KRANDICK‡

†*Computer and Information Sciences Department, University of Delaware, Newark, DE 19716, U.S.A.*
‡*Department of Mathematics and Computer Science, Drexel University, Philadelphia, PA 19104, U.S.A.*

Cylindrical algebraic decomposition requires many very time consuming operations, including resultant computation, polynomial factorization, algebraic polynomial gcd computation and polynomial real root isolation. We show how the time for algebraic polynomial real root isolation can be greatly reduced by using interval arithmetic instead of exact computation. This substantially reduces the overall time for cylindrical algebraic decomposition.

© 2002 Elsevier Science Ltd. All rights reserved.

1. Introduction

Cylindrical algebraic decomposition (CAD) (Collins, 1975; Caviness and Johnson, 1998) requires many very time consuming operations. These include resultant computation, polynomial factorization, algebraic polynomial gcd computation and isolation of the real roots of polynomials. Heretofore these operations have all been performed using exact integer arithmetic. In this paper we show that the root isolation can, almost always, be performed instead with interval arithmetic, drastically reducing the time for this operation.

In Section 2 we sketch the CAD method in order to show the role of root isolation, which occurs in the stack construction phase. This reveals that the problem is not just the isolation of the real roots of a single polynomial, but the isolation of all the real roots of a squarefree basis of algebraic polynomials.

In Section 3 we describe the process of algebraic polynomial basis real root isolation using the Descartes method (Collins and Akritas, 1976) and isolating interval refinement by bisection and evaluation, using exact arithmetic throughout. This includes description of the data representations that are used and the method of sign determination for algebraic numbers.

In Section 4 we describe the use of floating point interval arithmetic to perform algebraic polynomial basis real root isolation. We first attempt to do this with double precision hardware floating point arithmetic (Johnson and Krandick, 1997), which may fail due to either exponent limitation or mantissa limitation. When it fails we switch to a system of software floating point arithmetic with arbitrarily specified precision p, the number of words in the mantissa. We begin with precision 2 and repeatedly increase the precision until success is achieved or, since all precisions may fail in exceptional cases, until p exceeds some prescribed bound.

0747–7171/02/020145 + 13 $35.00/0 © 2002 Elsevier Science Ltd. All rights reserved.

In Section 5 we exhibit the performance of the floating point interval arithmetic method vs. the exact method in example applications. As examples we use the Solotareff approximation problem of degree 4 and the problem of computing the real solutions of a system of three polynomial equations in three variables.

2. Overview of CAD Construction

We shall be describing a program called **qepcad** (quantifier elimination by partial CAD). The original version of this program was written by Hoon Hong in 1990-91 and has subsequently been improved by several other persons, namely Christopher W. Brown, Mark J. Encarnación, Scott McCallum, and the authors of this paper. The program is based upon the SACLIB library of computer algebra programs (Collins *et al.*, 1993).

qepcad takes as input a formula in r variables, along with a specified ordering of these variables. In the following we will suppose that these variables, in their specified order, are $x_1, \ldots, x_r$. This formula has the form of a Boolean combination of atomic formulae, each of which is of the form $A(x_1, \ldots, x_r) = 0$ or $A(x_1, \ldots, x_r) > 0$, where each $A(x_1, \ldots, x_r)$ is an integral polynomial, that is, one with integer coefficients. This formula is possibly preceded by quantifiers in the variables $x_{k+1}, \ldots, x_r$, in that order. These are the variables that are to be eliminated, producing an equivalent formula in $x_1, \ldots, x_k$.

The first phase of the computation is called the normalization phase. This phase factors each of the polynomials in the input formula into irreducible polynomials and produces an equivalent formula, with the same quantifiers, involving only these irreducible factors. These irreducible polynomials are extracted, constituting a set $\mathcal{A}$.

The second phase is the projection phase. We need not go into the details of this phase. It produces a sequence of sets of polynomials $\mathcal{A}_r, \mathcal{A}_{r-1}, \ldots, \mathcal{A}_1$. The members of $\mathcal{A}_i$ are irreducible polynomials in $x_1, \ldots, x_i$, are of positive degree in x_i, and are called the level i projection factors. They are usually computed with the McCallum projection method (McCallum, 1998) which involves computing resultants and discriminants.

The third phase, the stack construction phase, is the focus of this paper. The projection factors from phase 2 determine a CAD of $\mathbb{R}^r$ into cells, in each of which each projection factor has invariant sign. However, due to the innovation of Collins and Hong (1991) we no longer compute this decomposition but instead a coarser CAD having the weaker but sufficient property that in each cell the input formula has invariant truth value.

We first compute a CAD of $\mathbb{R}$, using the level 1 projection factors. Then for $i = 2, 3, \ldots, r$ we use the level i projection factors (and the input formula) to compute a CAD of $\mathbb{R}^i$. The cells of the CAD of $\mathbb{R}^{i-1}$ are the bases of cylinders in $\mathbb{R}^i$. Each such cylinder is decomposed into a *stack* consisting of *sections* and *sectors*. $\mathbb{R}$ is itself regarded as a cylinder. Its sections are the one-point sets consisting of the real roots of the univariate level 1 projection factors. The sectors are the open intervals between consecutive roots and the semi-infinite intervals preceding and following all of the roots. If there are no real roots then the entire real line is a sector. Note that since the level 1 projection factors are irreducible they have no common roots.

The roots of the univariate projection factors are algebraic numbers. SACLIB and **qepcad** use two different representations of algebraic numbers, an *absolute representation* and a *relative representation*. The absolute representation of an algebraic number α consists of the unique univariate primitive irreducible integral polynomial $A(x)$ with positive leading coefficient having α as a root, called the *integral minimal polynomial*

of α, and an isolating interval I for α as a root of $A(x)$. I is a finite interval with rational endpoints containing α, but no other root of $A(x)$. The relative representation of an algebraic number β represents β as an element of some algebraic number field $\mathbf{Q}(\alpha)$. If $A(x)$ is the integral minimal polynomial of α and $\deg(A) = n$ then $\beta = B(\alpha)$ for a unique polynomial $B(x)$ in $\mathbb{Q}[x]$ with $\deg(B) < n$. If $B \neq 0$ then there exists uniquely a rational number b and a primitive integral polynomial $\bar{B}$ with positive leading coefficient such that $B = b \cdot \bar{B}$. The relative representation of β consists of b and $\bar{B}$, which are used instead of B in order to minimize the use of rational number arithmetic in computations, which is costly because of the integer gcd computations that are involved.

Construction of a CAD entails assigning to each cell an *index* and a *sample point*. The index of a cell is like an address that locates it relative to other cells. The index of a cell in $\mathbb{R}^i$ is an i-tuple of positive integers. In $\mathbb{R}$ the jth cell has index (j), where the cells are ordered by the "$<$" relation. In $\mathbb{R}^{i+1}$ the jth cell in the stack whose base has index $(k_1, \ldots, k_i)$ has index $(k_1, \ldots, k_i, j)$. The sample point of any cell is a point belonging to the cell. In $\mathbb{R}$ the sections are one-point cells whose sample points are therefore those unique points, algebraic numbers that are usually irrational. The sample points of the sectors are somewhat arbitrarily chosen rational numbers. In the following we will discuss selection of sample points in $\mathbb{R}^i$, $i > 1$.

Suppose that $(\alpha_1, \ldots, \alpha_i)$ is the sample point of a cell c in $\mathbb{R}^i$. We now wish to construct the stack having c as its base. Let $A_j(x_1, \ldots, x_{i+1})$, $1 \leq j \leq h$ be the level $i+1$ projection factors. We have a primitive element α such that $\mathbb{Q}(\alpha) = \mathbb{Q}(\alpha_1, \ldots, \alpha_i)$ with α in absolute representation and each α_l in relative representation as an element of $\mathbb{Q}(\alpha)$. We substitute each α_l for x_l in each A_j, obtaining univariate polynomials $\tilde{A}_j(x_{i+1})$ over $\mathbb{Q}(\alpha)$. We then compute a *squarefree basis* for these polynomials. This is a set of polynomials $B_1, \ldots, B_m$ such that each B_t is squarefree and is a divisor of some $\tilde{A}_j$, any two B_t's are relatively prime, and each $\tilde{A}_j$ is a power product of the B_t's. The squarefree basis is computed by a process of algebraic polynomial gcd computation using the method of Encarnación (1995).

Next we isolate the real roots of this squarefree basis of algebraic polynomials, the subject of this paper. The method used is to separately isolate the roots of each basis polynomial and then combine all the isolating intervals. But an isolating interval for one of the basis polynomials may overlap an isolating interval of another basis polynomial. Then we bisect the longer of the two intervals and retain the half that contains the isolated root. Repeating this process sufficiently many times will eventually produce two isolating intervals that are disjoint. In Section 3 we describe how exact arithmetic has been used for isolation and interval refinement. Then in Section 4 we describe how we now instead use floating point interval arithmetic.

The real roots of these basis polynomials define the sections of the stack that we are computing and the sectors are the regions (which are open connected sets) between consecutive sections, or preceding all sections, or following all sections. Let d be any cell in this stack. The first i coordinates of its sample point are the corresponding coordinates of c. If d is a section, the last coordinate is the basis polynomial real root that corresponds to the section. If d is a sector the last coordinate is a selected rational number between the last coordinates for the adjacent sections (or less than that for the first section, or greater than that for the last section).

There is more to CAD computation and quantifier elimination that need not be discussed in this paper. For example, each cell in the decomposition of $\mathbb{R}^k$, the space of the

free variables, must be assigned a truth value. This is also carried out during the stack construction phase. Following the stack construction phase comes the final phase, the solution formula construction phase, which attempts to construct a simple quantifier-free formula with desirable properties that is equivalent to the input formula. The latest work on this problem is due to Brown (1999).

3. Basis Real Root Isolation—Exact Arithmetic

Let A be an element of a squarefree basis of univariate polynomials. We determine an interval that is known to contain all real roots of A. Then we apply a recursive interval bisection procedure, the Descartes method (Collins and Akritas, 1976), to compute isolating intervals for A. We do this for every basis polynomial. The resulting isolating intervals are refined by further bisection until they are pairwise disjoint.

Both root isolation and refinement are particularly efficient if they operate on certain intervals we call "standard". We obtain standard intervals by starting with a standard interval and bisecting always at the midpoint.

Definition 3.1. A *binary rational number* is a rational number of the form $a2^k$ where a and k are integers. A *standard interval* is either a one-point interval $\{a\}$ where a is a binary rational number, or an open interval with binary rational endpoints of the form $(c2^k, (c+1)2^k)$ where c and k are integers.

3.1. root isolation

Our implementation of the Descartes method exploits the fact that standard intervals can be transformed onto the interval $(0, 1)$ using only shifts and integer arithmetic. Indeed, the interval $I = (c2^k, (c+1)2^k)$ can be transformed onto $(0, 1)$ through multiplication by 2^{-k} and subtraction of c. Under this bijective mapping, the roots of any polynomial $B(x)$ in I correspond to the roots of the polynomial $C = T_c(H_k(B))$ in $(0, 1)$ where H_k is the binary homothetic transformation which maps $B(x)$ to $\overline{B}(x) = B(2^k x)$, and T_c is the Taylor shift which maps $\overline{B}(x)$ to $C(x) = \overline{B}(x + c)$. The roots of C in $(0, 1)$ correspond, in turn, to the roots of $D = T_1(R(C))$ in $(0, \infty)$ where R is the reciprocal transformation which transforms the polynomial C into the polynomial $\overline{C}(x) = x^n C(1/x)$ where n is the degree of C. By Descartes' rule, finally, the polynomial D has no roots in $(0, \infty)$ if the coefficient sequence of D has no sign variations and exactly one root if there is exactly one sign variation; sign variations are counted after zeros are omitted from the coefficient sequence. The roots of C do not change if $\overline{B}$ is multiplied by a non-zero constant before T_c is applied. So, if B is an integral polynomial and k is negative, we multiply $\overline{B}$ by a power of 2 that makes the result integral and not divisible by 2; if k is positive, we divide by a power of 2 that makes the result non-divisible by 2.

The interval bisection process performed by the Descartes method can be represented by a binary tree where each node has an associated interval and an associated polynomial. The interval associated with the root of the search tree is a standard interval of the form $(0, 2^k)$ where 2^k is a root bound for the original polynomial A. The polynomial associated with the root of the search tree is $\overline{A} = H_k(A)$. Descartes' rule applied to $T_1(R(\overline{A}))$ determines either that A has no roots in $(0, 2^k)$, or that it has exactly one root in $(0, 2^k)$, or that the bisection process needs to be continued. To proceed from one node of the tree to its children, the method transforms the polynomial B of the parent node into

the polynomial $B_1(x) = H_{-1}B(x)$ of the left child. The polynomial of the right child is obtained by a Taylor shift, $B_2 = T_1(B_1)$. The intervals associated with the left and right child are the left and right halves, respectively, of the interval associated with the parent. Descartes' rule is applied to each of the polynomials $T_1(R(B_1(x)))$ and $T_1(R(B_2(x)))$. Eventually, isolating intervals for the positive roots of A are found. The negative roots of A are isolated as the positive roots of $A(-x)$.

The Taylor shift T_1 requires only additions in the coefficient domain; the binary homothetic transformation H_{-1} consists of multiplying the coefficients by powers of 2; the reciprocal transformation R, finally, is just an inversion of the coefficient sequence. It is clear how these operations are performed when the coefficients are elements of $\mathbb{Q}(\alpha)$ in relative representation.

The sign of a non-zero real algebraic number is obtained from its relative representation by the following non-trivial computation. Let $(b, \overline{B})$ be the relative representation of $\beta \in \mathbb{Q}(\alpha)$. Then b is a rational number and $\overline{B}$ is a primitive integral polynomial such that $\beta = B(\alpha)$ where $B = b \cdot \overline{B}$. Let $A(x)$ be the integral minimal polynomial of α and let I be an isolating interval for α as a root of A. The interval I is refined by bisection until the refined interval J does not contain any roots of $\overline{B}$. Then the sign of β can be computed as the product of the signs of b and $\overline{B}(c)$ where c is an arbitrary rational number in J. To test whether an interval contains a root of $\overline{B}$, Descartes' rule is applied to a transform of $\overline{B}$ as explained at the beginning of this section. If the transform has no variations, the interval contains no roots; otherwise, the bisection continues.

3.2. REFINEMENT

After isolating the real roots of each basis polynomial we transform the lists of disjoint isolating intervals into a single list of disjoint isolating intervals. To do this we merge two lists into one until only one list is left; to merge two lists we refine pairs of overlapping intervals.

As long as two standard isolating intervals of different lengths overlap we refine the longer interval by bisecting at the midpoint. When two overlapping standard intervals have the same length they are, in fact, equal. In this case, we refine each interval once—by bisecting at the midpoint. We terminate the procedure as soon as the refined intervals are disjoint.

DEFINITION 3.2. Let A be a squarefree polynomial, let α be a root of A, and let I be an isolating interval for α. Then the *trend* of I is the sign of A immediately to the right of α.

In our situation, trends are easily computed. The polynomial A is squarefree, and we have isolating intervals $I_1 < \cdots < I_m$ for all its real roots. If n is the degree of A and s is the sign of the leading coefficient, the trend of I_1 is s if n is odd, and $-s$ otherwise. The trend alternates from one interval to the next. When an interval is refined the trend stays the same.

When the trend of an isolating interval is known, each interval bisection requires just one polynomial sign evaluation. Indeed, if the sign at the bisection point equals the trend, the root is to the left of the bisection point; if the sign is zero, the root is the bisection point, and otherwise the root is to the right of the bisection point.

4. Basis Real Root Isolation—Floating Point Arithmetic

When exact integer arithmetic is used to isolate the real roots of a squarefree basis of algebraic polynomials, the most time-consuming polynomial transformations are the Taylor shifts T_c that occur in sign determination of algebraic numbers; also the Taylor shifts T_1 in the remainder of the Descartes method are expensive.

We present a method that avoids the Taylor shifts T_c completely and substantially reduces the computing time of the Taylor shifts T_1 and the homothetic transformations. The method uses interval arithmetic with floating point operations on the endpoints. Also real root refinement is sped up by using interval arithmetic. We assume the reader has a rudimentary acquaintance with interval arithmetic; otherwise he might, for example, consult the paper by Hickey *et al.* (2001).

The floating point computations almost always supply isolating intervals for the basis; there are, however, cases where they could terminate with a failure indication and exact integer computations would need to be used. The validated method and the exact method combined form an infallible algorithm.

The inputs to our algorithm are a real algebraic number α, a squarefree basis of polynomials over $\mathbb{Q}(\alpha)$, and a precision p. Let α be given by its integral minimal polynomial A, let n be the degree of A, and let I be the interval that isolates α as a root of A. We try to isolate the real roots of the basis using floating point computations of precision p.

4.1. conversion of the minimal polynomial

We embed A in a polynomial $\overline{A}$ with interval coefficients, that is, we embed each integer coefficient of A in an interval. The interval endpoints are floating point numbers that have a p-word mantissa and an exponent field of fixed size. Any integer a is either smaller than the smallest such floating point number, larger than the largest one, equal to a floating point number, or strictly in-between two consecutive floating point numbers. We compute a floating point interval of minimal width containing a—or determine that a is outside of the range of floating point numbers; the interval may be open or one-point.

The integer a is represented as a binary number, not necessarily normalized, and occupies one or more computer words. The floating point numbers have base 2 as well. We construct one interval endpoint directly from a and the other endpoint from the first.

The mantissa of the first endpoint consists of the leading bits of a; the exponent records the binary length of a. We want the mantissa to be normalized, so we determine the number of leading zero bits in the leading word of a; if needed, we shift the leading words of a to the left. The number of leading zero bits in a word is determined by a binary search algorithm that compares the word to certain powers of 2; in its last phase, the algorithm applies a linear search method.

When we compute the first endpoint, we also compute a *sticky bit* to remember whether the bits of a that did not become part of the mantissa were, in fact, all zero. The sticky bit tells us whether the first endpoint represents a exactly. In this case, the second endpoint is equal to the first; otherwise, the second endpoint is obtained by rounding the first, that is, by adding 1 in the last place. Which of the endpoints is left and which is right depends on the sign of a. Since we compute the sticky bit and the length of a, the conversion time is dominated by the sum of the mantissa length and the length of a.

4.2. conversion of the primitive element

Let $\overline{A}$ be the interval polynomial of precision p constructed in Section 4.1. Then $\overline{A}$ contains the minimal polynomial A of α. We want to refine the isolating interval I of α using bisections and polynomial sign evaluations. By Section 3, I is standard; hence the midpoint a of I is a binary rational number. We convert a to a floating point number $\overline{a}$ whose mantissa is so long that the conversion is exact. Then we try to evaluate the sign of $\overline{A}$ at $\overline{a}$ using floating point arithmetic.

Sign evaluation is performed with Horner's scheme. Using interval arithmetic would require n interval additions and n interval multiplications, thus $2n$ floating point additions and, usually, $2n$ floating point multiplications. Since only the sign of the result is needed, we can improve on this. If $\overline{a} > 0$, we evaluate $\overline{A}(x)$ at $x = \overline{a}$ with rounding directed down, using only the left endpoint of each coefficient of $\overline{A}$. We thereby obtain a lower bound for the exact value of $A(a)$. If the lower bound is positive then $A(a) > 0$. If $\overline{a} < 0$ we instead evaluate $\overline{A}(-x)$ at $-\overline{a}$. If the lower bound is negative, then we compute an upper bound for $A(a)$ in a similar manner, rounding up. If the upper bound is negative then $A(a) < 0$. Otherwise the sign determination fails; higher precision is required. By this method we reduce the average number of additions and multiplications, each, to $3n/2$.

The precision needed to represent the binary rational number a exactly as a floating point number will often be less than p. Hence, we use a multiplication routine that efficiently multiplies two numbers of different precisions; the product has the greater of the two precisions.

We continue the bisections and sign evaluations until the interval I is refined to some small specified width—but at most as far as can be achieved with precision p.

Let I' be the refinement of I that we obtain. We embed I' in the smallest p-precision floating point interval I'' containing I'. The endpoints of I'' are obtained by rounding the binary rational endpoints of I' outwards. Binary rationals are rounded by rounding their numerators using the techniques from Section 4.1.

4.3. conversion of the algebraic polynomials

Let B be a polynomial of the squarefree basis. The coefficients of B are of the form $C(\alpha)$ where C is a rational polynomial represented by a pair (r, D) where r is a rational number and D is an integral polynomial of degree $< n$ such that $C = r \cdot D$.

We embed the coefficients of $r \cdot D$ in floating point intervals as follows. If r is represented as the quotient of integers s and t, $t \neq 0$, and if d is an integer coefficient of D, then we embed sd into an interval and t into an interval that does not contain 0; we form the quotient of the two intervals to obtain an interval containing rd. By doing this for all the coefficients we obtain an interval polynomial $\overline{C}$ containing C.

Let I'' be the floating point interval containing α from Section 4.2. By evaluating $\overline{C}$ at I'' using interval arithmetic we obtain an interval containing $C(\alpha)$. By doing this for all the coefficients C of B we obtain an interval polynomial $\overline{B}$ containing B.

4.4. the interval descartes method

Let B be a basis polynomial and let $\overline{B}$ be the interval polynomial containing B that was constructed in Section 4.3. We try to isolate the real roots of B by subjecting $\overline{B}$ to an interval version of the Descartes method; we perform the Descartes method but

replace the exact polynomial transformations by the corresponding interval operations. In this way, the time consuming operations required to compute the sign of an algebraic number β are replaced by the (trivial) sign computation of an interval containing β.

Since the exact polynomial B is contained in the input polynomial $\overline{B}$, any interval coefficient of any interval transform of $\overline{B}$ contains the corresponding scalar coefficient of the corresponding exact transform of B. The Descartes method does not require the interval coefficients to be particularly narrow—as long as it can decide whether the number of coefficient sign variations of certain transforms of the input polynomial is > 1, $= 1$, or $= 0$. Sometimes, this decision is possible even in the presence of undefined signs; indeed, the sequence $(+, ?, -)$ has exactly one sign variation—no matter what sign is substituted for the undetermined sign in the middle.

If the interval Descartes method can decide all questions about coefficient sign variations that arise for some input polynomial, it will construct the same search tree as the exact Descartes method—and produce the same isolating intervals. If the number of coefficient sign variations cannot be determined at some node, the tree stops growing at that node.

The search tree built in this way by the interval Descartes method is a subtree of the tree the exact method would build for any scalar polynomial contained in the input polynomial. In particular, since the input polynomial $\overline{B}$ contains the squarefree polynomial B, the interval Descartes method will build a finite tree. More generally, the tree will be finite whenever at least one coefficient of the input polynomial has non-zero width.

If the interval Descartes method does not completely reveal the true search tree for B at precision p, one may re-run the computation using a higher precision; there are, however, input polynomials for which the interval Descartes method does not succeed—regardless of the precision. Indeed, let α be any real algebraic number in the interval $(0, 1/3)$ of degree > 1, and let $B(y) = (\alpha + 1)y^2 + (-2\alpha)y + \alpha$. When the algorithm is called for an interval approximation of $B(y)$, it computes 1 as a bound for the positive roots and applies the transformation $T_1 R$ using interval arithmetic. This results in an interval approximation to the polynomial $\alpha y^2 + 0y + 1$. The sign of the interval coefficient of y is indeterminate because, for any precision, the interval will contain 0 in its interior.

The most time consuming polynomial transformations of the interval Descartes method are the Taylor shifts T_1. The computing time for T_1 is codominant with $n^2 p$ since T_1 consists of $n(n+1)/2$ additions of intervals with floating point endpoints of precision p. It is thus important to use an efficient method (Collins and Krandick, 2000) for floating point addition. The computing time of a binary homothetic transformation H_{-1} is codominant with n since H_{-1} is performed by decrementing the fixed-size exponents in n floating point intervals.

4.5. REFINEMENT AND MERGE SORT

The isolating intervals for the basis polynomials are merged as in exact computation (Section 3.2). The only difference is that the refinement of overlapping intervals now uses floating point arithmetic to evaluate the sign of a polynomial at a bisection point. This is done using the techniques of Section 4.2. If two overlapping intervals cannot be separated at precision p the algorithm returns with a failure indication.

4.6. PRECISION MANAGEMENT

Sections 4.1– 4.5 describe a method for algebraic polynomial basis real root isolation that can be applied for various choices of the precision p. For a given polynomial basis we invoke the method first with hardware provided floating point arithmetic. The *double* format of IEEE-754 arithmetic (IEEE, 1985, 1987) provides 53 bits for the mantissa (including the hidden bit) and an 11-bit field for the exponent. This suffices in many cases to produce disjoint isolating intervals for all the real roots of the basis. In some cases, however, the fields provided for the mantissa or the exponent are too short. If the method fails due to exponent overflow or underflow, we re-start it using software supported floating point arithmetic with $p = 2$ computer words for the mantissa (58 bits in our implementation) and a whole word (32 bits) for the exponent field. In other cases of failure we re-run the method using $p = 3$ computer words for the mantissa. For all realistic inputs, a one-word exponent will suffice. If basis root isolation fails for p mantissa words, we re-run the method with $p + 1$ mantissa words. As pointed out in Section 4.4, there are cases where the exact method must be used. In our applications, such cases did not arise and the maximum required precision was $p = 6$. So, one may define, somewhat arbitrarily, the maximum precision that should be employed by the floating point method as $p_{\max} = 10$.

4.7. EXPONENT LIMITATION

We use hardware provided floating point arithmetic in such a way that the execution of our programs remains stable when an exponent field overflows or underflows as a result of a floating point operation. In these cases, an exception handler raises a global flag, but the computation continues and returns a result. By checking the global flag, high-level routines can decide whether the result has been corrupted. After this decision the global flag is cleared.

We do not provide any similar mechanism for software supported floating point arithmetic since any exponent arising in practice will fit into one computer word.

5. Examples

To show the effectiveness of our methods in CAD-based quantifier elimination we use two example applications. The first example is the problem of computing the real solutions of a system of three polynomial equations in three variables; the second example is the Solotareff approximation problem of degree 4 (Achieser, 1956). In both examples, the time to isolate the real roots of all occurring squarefree polynomial bases was reduced by several orders of magnitude. Hardware IEEE-double precision floating point arithmetic sufficed for the polynomial system. For Solotareff's problem, however, several IEEE-double precision computations failed and it was necessary to use software-based floating point numbers with up to 6 mantissa words to complete all computations.

All experiments were performed using version 18 of **qepcad** built with version 2.1 of SACLIB. All timings were obtained on a SUN4U/400 Ultra-450 with a 400 MHz CPU and 2GB of memory. SACLIB and **qepcad** were compiled using version 5.0 of Sun's C compiler.

Table 1. Polynomial equation system computing times (in seconds).

	Real Root Isolation			Garbage	Total
	Exact arith.	Hardware arith.	Software arith.	collection	time
Old	0.57			0.11	1.26
New		0.005		0.04	0.62

5.1. SYSTEM OF POLYNOMIAL EQUATIONS

In this example a partial CAD was constructed in order to determine the real solutions of a system of three polynomial equations in three variables with each polynomial of total degree 2 and random 5-bit integer coefficients. Equational constraints (see McCallum, 1999) were used along with the variable ordering (x, y, z). One million computer words were allocated for list cells and the threshold for data base entries was set at 10 ms. The input formula was the following:

$$[-12z^2 - 3yz + xz - 27z - 4y^2 - 11xy - 5y + 29x^2 + 11x - 27 = 0$$
$$\wedge -25z^2 - 23yz + 23xz + 4z + 2y^2 + 7xy + 21y + 4x^2 - 15x - 30 = 0$$
$$\wedge -14z^2 + 27yz - 29xz + 11z + 4y^2 - 31xy + 22y - 12x^2 - 28x - 9 = 0].$$

Nine stacks were constructed, consisting of 109 cells. The command d-true was then used to display the solutions to 10 decimal places. There were two solutions:

$$(1.4147645223, -6.2194933011, 1.0268880927)$$
$$(-0.9142772570, 0.3973629958, -0.4932964552).$$

Table 1 shows the computing times for this problem when exact arithmetic was used ("Old") and again when interval arithmetic was used ("New"). Garbage collection times are shown because the use of interval arithmetic reduces the use of list storage, and thereby the time required for garbage collection. Garbage collection time is included in total computing time, but not in the times for root isolation. In this problem hardware interval arithmetic sufficed for all root isolations. Note that root isolation time was reduced by a factor of more than 100, and total time was reduced by a factor of slightly more than 2.

5.2. THE SOLOTAREFF PROBLEM

In this example, **qepcad** was used to solve the Solotareff approximation problem of degree 4. The general problem, known as Solotareff's first problem, is to find the best approximation, in the uniform norm, on the interval $[-1, +1]$, of a polynomial of degree n by a polynomial of degree $n-2$ or less. Equivalently, the polynomial to be approximated can be just the binomial $x^n + rx^{n-1}$. We then seek to obtain each coefficient of the best approximation as an algebraic function of r. In the case $n = 4$, we take the best approximation to be $ax^2 + bx + c$. Using the definition of best uniform appproximation, we could then formulate the problem of finding a as a function of r as

$$(\exists b)(\exists c)(\forall d)(\forall e)(\forall f)(\forall x)(\exists y)[-1 \leq x \wedge x \leq 1 \Rightarrow -1 \leq y \wedge y \leq 1$$
$$\wedge((x^4 + rx^3) - (ax^2 + bx + c))^2 \leq ((y^4 + ry^3) - (dy^2 + ey + f))^2].$$

However, this problem, with 9 variables, is far too difficult for **qepcad**. Instead, we utilize some theorems which can be found in Achieser's book (Achieser, 1956). By a theorem of Chebyshev, the unique polynomial $P(x)$ of degree m that is the best uniform approximation to a continuous function $f(x)$ on a finite interval $[a, b]$ is characterized by the existence of $m + 2$ consecutive points at which $|f(x) - P(x)|$ assumes its maximum value on the interval, with the sign of $f(x) - P(x)$ alternating. Let $P(x)$ be the best approximation to $x^n + rx^{n-1}$ on the interval $[-1, 1]$. Replacing x with $-x$, and multiplying by $(-1)^n$, the polynomial $(-1)^n P(-x)$ is the best approximation to $x^n - rx^{n-1}$. Therefore we may assume that $r > 0$. Achieser shows (with slightly different notation) that for $0 \leq r \leq n \tan^2 \pi/2n$, the best approximation to $x^n + rx^{n-1}$ is $x^n + (-1)^n rx^{n-1} - T_n((-x - r/n)/(1 + r/n))$, where $T_n(x)$ is the Chebyshev polynomial $\cos(n \arccos x)$. For the case $n = 4$, this gives us the following:

$$
\begin{aligned}
a &= 1 + 1/2r - 5/16r^2, \\
b &= 1/2r + 1/4r^2 - 1/32r^3, \\
c &= -1/8 - 1/8r + 1/64r^2 + 3/128r^3 + 1/2048r^4.
\end{aligned}
$$

For $n = 4$, $n \tan^2 \pi/2n = 4 \tan^2 \pi/8 = 12 - 8\sqrt{2}$, the least root of $x^2 - 24x + 16$.

Achieser goes on to show that for $r \geq n \tan^2 \pi/2n$, two of the n consecutive extremum points are -1 and 1. The polynomial $x^n + rx^{n-1} - P(x)$ must be positive at $x = 1$, and so, for even n, it must be negative at -1. Let $E(x) = x^4 + rx^3 - ax^2 - bx - c$. Then $E(1) = -E(-1)$, from which we obtain $c = 1 - a$. Let the intermediate extremum points be u and v, $u < v$. Then $E(u) = E(1)$, from which $u^4 + ru^3 - au^2 - bu - c = 1 + r - a - b - c$. Simplifying and factoring, $(u-1)(u^3 + ru^2 + u^2 - au + ru + u - b - a + r + 1) = 0$ and clearly $u - 1 \neq 0$. Similarly, from $E(v) = E(-1)$ we obtain $v^3 + rv^2 - v^2 - av - rv + v - b + a + r - 1 = 0$. Let $E'(x) = 4x^3 + 3rx^2 - 2ax - b$. Then $E'(u) = E'(v) = 0$. Also, $r - b = E(1) > 0$. This inessential inequality somewhat reduces the CAD computations. Finally, since r is greater than the least root of $x^2 - 24x + 16$ if and only if either $r^2 - 24r + 16 < 0$ or $r > 1$, we have arrived at the following formulation of Solotareff's first problem for degree 4, for obtaining a as a function of r.

$$
\begin{aligned}
&(\exists b)(\exists u)(\exists v)[[r^2 - 24r + 16 < 0 \vee r > 1] \\
&\wedge\ -1 < u \wedge u < v \wedge v < 1 \wedge r - b > 0 \\
&\wedge u^3 + ru^2 + u^2 - au + ru + u - b - a + r + 1 = 0 \\
&\wedge v^3 + rv^2 - v^2 - av - rv + v - b + a + r - 1 = 0 \\
&\wedge 4u^3 + 3ru^2 - 2au - b = 0 \wedge 4v^3 + 3rv^2 - 2av - b = 0].
\end{aligned}
$$

qepcad was applied to this formula with the variable ordering (r, a, b, u, v), with equational constraints, and with 4 000 000 words allocated for list cells. The projection phase of the computation revealed the existence of just one equational constraint polynomial at level 2, which consisted of seven irreducible factors. By inspection we guessed that the solution corresponded to just a particular one of these. Using a new extension of **qepcad**'s selected-cells-condition command, we directed **qepcad** to construct stacks in 3-space only over sections of this factor. The correctness of our guess was confirmed by the solution that was obtained, since we know that best uniform approximations are unique.

Table 2 shows the computing times for this problem, revealing that the time for root isolations was reduced by a factor of more than 100 and the total time was reduced by 46%. In this computation there were 206 applications of polynomial basis real root isolation using hardware IEEE double precision floating point numbers, and the method failed

Table 2. Solotareff problem, solution for a, computing times in seconds.

	Real Root Isolation			Garbage	Total
	Exact arith.	Hardware arith.	Software arith.	collection	time
Old	24.5			10.1	66.4
New		0.088	0.139	3.8	35.7

Table 3. Solotareff problem, solution for b, computing times in seconds.

	Real Root Isolation			Garbage	Total
	Exact arith.	Hardware arith.	Software arith.	collection	time
Old	571			436	3577
New		0.113	0.659	245	2770

in just 13 of these applications. In these 13 cases, software multiprecision floating point arithmetic of precision 2 was applied, and 3 cases were completed successfully. In the remaining 10 cases, precision 3 was applied, and 7 cases were completed successfully. In the remaining 3 cases precision 4 succeeded.

The solution formula produced by **qepcad** using the command "solution E" is the following.

$$\begin{aligned} & r > \mathrm{root}_1[r^2 - 24r + 16] \\ \wedge\, a = \mathrm{root}_{-1}[&324a^4 + 324r^2a^3 - 2016a^3 + 108r^4a^2 - 1128r^2a^2 + 4576a^2 \\ &+12r^6a - 224r^4a + 1392r^2a - 4480a - 15r^6 + 112r^4 - 608r^2 + 1600]. \end{aligned}$$

The notation root_1 in this formula means the first real root (of the argument polynomial) beginning with the smallest, in other words, the smallest real root; likewise, root_{-1} designates the first real root beginning with the largest, that is, the largest real root. For every real number r greater than the smallest real root of $x^2 - 24x + 16$, the coefficient a, as a function of r, is the largest real root of the displayed polynomial in r and a. We know already that the coefficient c is simply $1 - a$.

To compute the coefficient b we called **qepcad** for the same formula as for a but with the variable ordering changed to (r, b, a, u, v) and the quantifiers changed to $(\exists a)(\exists u)(\exists v)$; we allocated 8 000 000 computer words for list cells. Table 3 shows the computing times for this problem. In this computation there were 154 applications of polynomial basis real root isolation using hardware floating point basis isolation, and this failed in 17 cases. Of these, software arithmetic of precision 2 succeeded in 4 cases, precision 3 sufficed for 5 cases, precision 4 sufficed for 6 cases, and precision 6 was required for the remaining 2 cases.

The following solution formula was obtained.

$$\begin{aligned} & r > \mathrm{root}_1[r^2 - 24r + 16] \wedge P(r, b) \le 0 \\ \wedge[&[Q(r) \le 0 \wedge b \le \mathrm{root}_1(P(r, b))] \vee [Q(r) \ge 0 \wedge b \ge \mathrm{root}_{-1}(P(r, b))]] \end{aligned}$$

where

$$\begin{aligned} P(r, b) = {} & 78732b^4 + 8748r^3b^3 - 291600rb^3 + 324r^6b^2 - 18684r^4b^2 \\ & +403272r^2b^2 - 1024b^2 + 4r^9b - 616r^7b + 12692r^5b - 246800r^3b \\ & +2048rb - 3r^{10} + 280r^8 - 2708r^6 + 56332r^4 - 1024r^2 \end{aligned}$$

and

$$Q(r) = 243r^8 - 8532r^6 - 15920r^4 + 2624r^2 - 1024.$$

A separate **qepcad** application with input formula $[P(r,b) = 0]$, taking only 62 ms, revealed that $Q(r)$ is a factor of the discriminant of $P(r,b)$, that the discriminant of $P(r,b)$ has only one positive real root, a root, α, of $Q(r)$ that is approximately 6.0728651867, and that, for $r > 0$, $P(r,b)$ has two real roots, which coincide at α. Since the leading term of $P(r,b)$ is positive, it follows that $P(r,b) \leq 0 \wedge b \leq \text{root}_1(P(r,b))$ is equivalent to $b = \text{root}_1(P(r,b))$, and likewise $P(r,b) \leq 0 \wedge b \geq \text{root}_{-1}(P(r,b))$ is equivalent to $b = \text{root}_{-1}(P(r,b))$. So b is the first root of $P(r,b)$ for $r \leq \alpha$, the second root of $P(r,b)$ for $r > \alpha$.

In the computation of b nearly all of the time is taken by less than a dozen of the stack constructions. These involve a minimal polynomial of degree 36 that has coefficients up to 395 bits long.

Acknowledgement

The authors acknowledge partial support by NSF Grant 9712246 (G.E.C.) and DFG project SFB 376 (W.K.).

References

Achieser, N. I. (1956). *Theory of Approximation*, New York, Frederick Ungar Publishing Co.

Brown, C. W. (1999). Guaranteed solution formula construction. In Dooley, S. ed., *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, pp. 137–144. ACM Press.

Caviness, B. F., Johnson, J. R. (eds) (1998). *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Springer.

Collins, G. E. (1975). Quantifier elimination for the theory of real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages, 2nd GI-Conference*, LNCS, **33**, pp. 134–183. Berlin, Springer (reprinted with corrections by the author as pp. 85–121 in Caviness, B. F., Johnson, J. R., eds (1998)).

Collins, G. E., Akritas, A. G. (1976). Polynomial real root isolation using Descartes' rule of signs. In Jenks, R. D. ed., *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, pp. 272–275. ACM Press.

Collins, G. E., Hong, H. (1991). Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, **12**, 299–328 (reprinted as pp. 174–200 in Caviness, B. F., Johnson, J. R., eds (1998)).

Collins, G. E. *et al.* (1993). `SACLIB` User's Guide. Technical Report 93-19, Research Institute for Symbolic Computation, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria.

Collins, G. E., Krandick, W. (2000). Multiprecision floating point addition. In Traverso, C. ed., *International Symposium on Symbolic and Algebraic Computation*, pp. 71–77. ACM Press.

Encarnación, M. J. (1995). Computing gcds of polynomials over algebraic number fields. *J. Symb. Comput.*, **20**, 299–313.

Hickey, T., Ju, Q., van Emden, M. H. (2001). Interval arithmetic: from principles to implementation. *J. ACM*, **48**, 1038–1068.

IEEE (1985). *ANSI/IEEE Standard 754-1985 for Binary Floating-point Arithmetic*, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA.

IEEE (1987). ANSI/IEEE Standard 754-1985 for binary floating-point arithmetic. *ACM SIGPLAN Notices*, **22**, 9–25. (Reprint of IEEE (1985).)

Johnson, J. R., Krandick, W. (1997). Polynomial real root isolation using approximate arithmetic. In Küchlin, W. ed., *International Symposium on Symbolic and Algebraic Computation*, pp. 225–232. ACM Press.

McCallum, S. (1998). An improved projection operation for cylindrical algebraic decomposition. In Caviness, B. F., Johnson, J. R. eds, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pp. 242–268. Springer.

McCallum, S. (1999). On projection in cad-based quantifier elimination with equational constraint. In Dooley, S. ed., *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, pp. 145–149. ACM Press.

Received 3 May 2001
Accepted 9 May 2002